## AUTHOR

Name - Aarav Shukla
Roll no. - 24f2004213
Student email - 24f2004213@ds.study.iitm.ac.in
Hello, My name is Aarav Shukla and I am pursuing the B.S in Data Science and Applications from IIT Madras. I am currently at Diploma level. I have very much interest in AI/ML/DL and Web Development. I always try to get better at coding.

## DESCRIPTION

This project is a **quiz management application** that allows users to attempt quizzes, track their performance, and view statistical analysis. Admins can create subjects, chapters, and quizzes while monitoring users' progress through a performance dashboard.

## TECHNOLOGIES USED

1. Flask
2. Flask-SQLAlchemy
3. Flask-Login
4. SQLite
5. Bootstrap
6. Chart.js
7. Jinja2
8. Matplotlib
9. Numpy
10. SQLAlchemy Functions (func module)
11. Flask Flash Messaging
12. Base64 & io

Purpose behind Technologies choices-

1. **Flask:** Lightweight and easy-to-use web framework.
2. **Flask-Login:** Provides session management and user authentication without manual handling.
3. **Flask-SQLAlchemy:** Simplifies database interactions and prevents SQL injection risks.
4. **Matplotlib & NumPy:** Used for data visualization and statistical computations for performance analytics.
5. **SQLite:** A simple and efficient database for small-scale applications like this.
6. **Base64 & io:** Allows embedding images (like graphs) directly into HTML templates.

## DB SCHEMA DESIGN

**User Table**

- `user_id` (PK, Auto-increment) – Unique user ID.
- `user_email` (Unique, Not Null) – Stores user email.
- `password`, `user_name`, `user_qualification`, `dob` (Not Null) – Essential user details.

**Role & UserRole Tables**

- `Role`: `id` (PK, Auto-increment), `name` (Unique, Not Null) – Defines user roles.
- `UserRole`: Links users and roles using `user_id` & `role_id` (FKs).

**Subject & Chapter Tables**

- `Subject`: `id` (PK, Auto-increment), `name` (Unique, Not Null), `description`.
- `Chapter`: `id` (PK), `name` (Not Null), `description`, `subject_id` (FK).

**Quiz Table**

- `id` (PK, Auto-increment), `title` (Not Null), `chapter_id` (FK), `date_of_quiz`, `time_duration` (>0), `time_start`, `remarks`.

**Question Table**

- `id` (PK, Auto-increment), `quiz_id` (FK), `question_statement` (Not Null).
- `option1-4` (Not Null), `correct_option` (1-4).

**Score Table**

- `id` (PK, Auto-increment), `quiz_id` (FK), `user_id` (FK).
- `time_stamp_of_attempt` (Default: CURRENT_TIMESTAMP), `total_scored` (≥0).

**Reason for designing this way-**

- **Normalization** prevents redundancy.
- **Constraints** (NOT NULL, UNIQUE, CHECK) ensure data integrity.
- **Foreign Keys** maintain referential integrity.
- **Scalability** supports future features.

**API Design-**

**Authentication & Authorization**

- **Login**: `/login`
- **Logout**: `/logout`
- **Register**: `/signup`

**User Routes**

- **Get User Dashboard**: `/user_dashboard`
- **Get Admin Dashboard**: `/home`
- **Get Application Landing page**: `/`
- **Get User Scores**: `/scores`
- **Get user profile:** `'/user_profile/<int:user_id>', methods=['GET']`
- **Get User Result for Quiz**: `/result/<int:quiz_id>`

**Subject Routes**

- **Api for subject:** `'/api/subjects', methods=['GET']`
- **Add New Subject**: `'/subject', methods=['GET', 'POST']`

**Chapter Routes**

- **API for chapters:** `'/api/chapters', methods=['GET']`
- **Add New Chapter**:`'/chapter/<int:subject_id>', methods=['GET', 'POST']`
- **Edit Chapter**: `'/edit_chapter/<int:chapter_id>', methods=['GET', 'POST']`
- **Delete Chapter**: `'/delete_chapter/<int:chapter_id>', methods=['POST']`

**Quiz Routes**

- **Get All Quizzes**: `'/quiz-list', methods=['GET', 'POST']`
- **Add New Quiz**: `/add-quiz`
- **Get Quiz Details**: `/quiz/details/<int:quiz_id>`
- **API for quizzes:** `'/api/quizzes', methods=['GET']`
- **User Attempt Quiz**: `/quiz/<int:quiz_id>`

**Question Routes**

- **Add Question to Quiz**: `'/add-question/<int:quiz_id>', methods=['GET', 'POST']`
- **Edit Question**: `'/edit_question/<int:question_id>', methods=['GET', 'POST']`
- **Delete Question**: `'/delete_question/<int:question_id>', methods=['POST']`

**Score Routes**

- **API for scores:** `'/api/scores', methods=['GET']`
- **Get All Scores**: `/scores`

**Search Route**

- **Search**: `'/search', methods=['GET']`
    - 
    - `filter` values: `users`, `subjects`, `quizzes`, `questions`

**Summary Route**

- **Admin Summary Graphs:** `'/admin/summary'`
- **User Summary Graphs**: `'/performance'`

**Architecture and Features**

The application follows an MVC-like structure where controllers handle the logic, templates manage the UI, and a database stores persistent data. The `controller/` directory contains key files like `auth_routes.py` for authentication, `routes.py` for general routing, and `database.py` for database

management. The `templates/` folder stores HTML templates rendered with Jinja2, while `static/` contains stylesheets and other static assets. The main entry point, `main.py`, initializes the Flask application, loads configurations from `config.py`, and registers routes.

- **Default Features:**
    - User authentication (handled in `auth_routes.py`).
    - Quiz management (handled in `routes.py`).
    - Database storage using SQLite (`database.py`).
    - Jinja2-based rendering for dynamic templates.
- **Additional Features:**
    - Custom styling through `static/styles.css`.
    - Persistent storage using `instance/database.sqlite3`.
    - Separation of concerns via a modular `controller/` directory.
    - Configuration management with `config.py`.

**Video**

2025-03-12 17-56-51.mkv